COMPUTER RECREATIONS

How close encounters with star clusters are achieved with a computer telescope

by A. K. Dewdney

Deep in space, a star cluster performs a cosmic dance to the tune of gravity. During a human lifetime the stars barely move; over a longer span, in which years are equivalent to seconds, they trace out a tangled figure of orbits. Occasionally a single star encounters a neighbor in a pas de deux that hurls it out into space. If such escapes are more than occasional, the cluster gradually shrinks and the core begins to collapse.

A powerful telescope can reveal the structure of some clusters in our galaxy but it cannot compress years into seconds—only a computer is able to do that. A computer can also be programmed to serve as a kind of telescope for viewing hypothetical clusters. At cosmic speed one can watch the movement of the members of a cluster as a succession of snapshots in which each star leaves a dotted trail that weaves through the cluster [*see illustration on opposite page*].

Do gravitational forces alone account for the evolution that astronomers infer from observed clusters? Computers help to find answers to this question and related ones. A conference of simulators and theoreticians met at Princeton University in May, 1984, to discuss the consistency of hypothetical and actual star clusters. It was the 113th symposium of the International Astronomical Union; the entire symposium was devoted to the dynamics of star clusters.

It is relatively easy to choreograph a cosmic ballet. In principle the stellar interactions within a cluster are classically simple: both members of a pair of stars experience a gravitational force that is proportional to the inverse square of the distance between them. The force is also proportional to the product of the two stellar masses. Such a formula is easy to compute: multiply the masses together; then multiply the product by a constant of proportionality and divide by the square of the distance between the two stars. The sum total of all such paired forces acting through time presumably determines the pattern of movement within the cluster. A program, called CLUSTER, computes the sum of the forces for each star and moves the sum from its present position to a new one nearby. It does this repeatedly during centuries of simulated time.

A certain tedium attends typing in the coordinates and velocities of many stars, but once this is done an armchair universe unfolds on the display screen. Stars at the center of the cluster follow wobbly, erratic courses; those at the periphery drift away, stop and then glide back. The most interesting events include close encounters and escapes.

When two stars approach each other closely, they impart a tremendous gravitational boost to each other and speed apart. Escapes are usually the result of one or more close encounters. When a star speeds away from its cluster, there are only two possibilities: either the star returns or it does not. An astronomical body has an escape velocity that depends on its mass and on the mass of the body or object from which it escapes. If the velocity is attained by a star moving outward from its cluster, it will never return. Inexperienced cluster buffs are likely to witness frequent escapes from the configurations they design. In fact, a common initial experience is to see one's hoped-for dance disintegrate. It is wise to practice by building a system of two or three stars.

The structure of the CLUSTER program is simple. It consists of an initialization loop followed by a double loop. Within the double loop the acceleration, velocity and position of each star are updated according to the summed attractions of the other stars. I shall describe a particularly simple version of the program in which the time increment, force constant and stellar masses are all built in. In spite of its simplicity, however, this version of CLUSTER seems capable of simulating almost the entire range of cluster behavior. Three sets of arrays are used. The first set keeps track of the accelerations currently experienced by the stars in each of three coordinate directions. The arrays are called *ax*, *ay* and *az*. Thus *ax*(*i*), *ay*(*i*) and *az*(*i*) indicate the *x*, *y* and *z* components of the *i*th star's acceleration. The contents of the three arrays alone do not need to be initialized at the start of the program. The second set of arrays, *vx*, *vy* and *vz*, define velocities: *vx*(*i*), *vy*(*i*) and *vz*(*i*) register the *x*, *y* and *z* components of the *i*th star's velocity. The third set of arrays record positions: *x*(*i*), *y*(*i*) and *z*(*i*) are the *i*th star's *x*, *y* and *z* coordinates of position. The starting values for the arrays *x*, *y*, *z* and *vx*, *vy*, *vz* must be initialized at the head of the program.

The main body of the CLUSTER program follows the initialization segment. The double loop can be entered and reentered endlessly, or the programmer can establish the specific conditions that control reentry. The outer loop considers each star in turn and sets the acceleration components to zero. After this has been done the inner loop computes the forces produced on each star by its companions in the cluster.

For example, let us assume that the index of the outer loop is *i* and that the inner-loop index is *j*. The inner loop first checks to determine whether *i* is equal to *j*. If it is, the program does not invoke the force computation: a star does not attract itself. In any event, to compute force under the circumstance would cause the computer to attempt division by zero. (This is the only situation that can actually make me feel sorry for a computer.) When *i* and *j* are not equal, CLUSTER uses Euclid's formula for distance between the stars: the differences of the *x*, *y* and *z* coordinates are squared and added together. The result, of course, is the square of the distance. Next, the inner loop tests whether this number is 0. If it is, an alarm of some kind should be raised because the computer is about to be asked to divide by zero. My version of the program prints "Collision!"

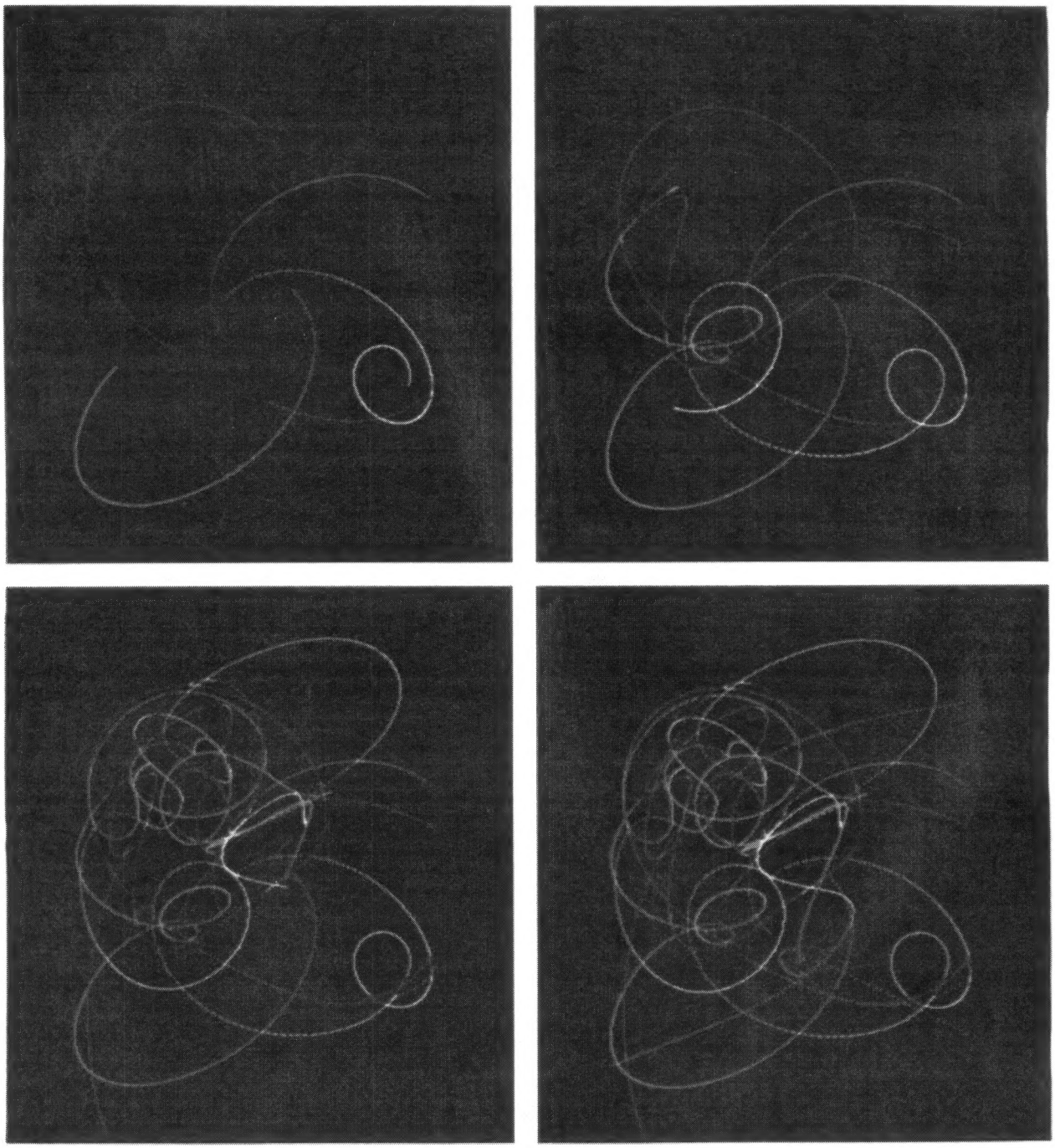
If nothing is amiss, the inner loop computes the distance between the stars by taking the square root *d* of the squared distance computed earlier. It then divides 1,000 by the square of the distance, a calculation that yields the force. The final task to be performed within the inner loop is to determine the acceleration components of the

ith star. This value is obtained by adding together the force contributions from the other stars. For example, the x component of acceleration can be written generically as follows:

$$ax(i) \leftarrow ax(i) + f \times (x(j) - x(i))/d$$

Here f and d represent the force and distance. The ratio of the x distance between the ith and jth stars to the total distance is precisely the fraction of the force that acts on the ith star in the x direction. The y and z components of acceleration are computed by analogous formulas.

Two more loops, one following the other, complete the program. The first updates velocity and the second updates position. There is a subtle point here, first brought to my attention by John H. Hubbard, the Cornell University mathematician whose advice on computing the Mandelbrot set was eminently useful [see "Computer Recreations," August, 1985]. It is indeed possible to compute position before



Four stars put on a cosmic ballet for a few years and then leave the stage

computing velocity without producing strange-looking results. Yet the motions of the stars would in time become strangely wrong, because such an operation would violate the law of energy conservation.

The velocity-updating loop merely adds acceleration to velocity, according to the following formula:

$$vx(i) \leftarrow vx(i) + ax(i)$$

Here it is assumed that the time increment equals the time unit in which velocity is expressed. The same kind of formula is used to calculate *vy* and *vz*. The position calculations done in the final loop are equally simple:

$$x(i) \leftarrow x(i) + vx(i)$$

The entries of the *y* and *z* arrays are similarly updated. Drawing on the information from the final loop, CLUSTER places each point on the two-dimensional surface of the display screen. It does so by plotting the first two position coordinates while suppressing the third. The natural result of this arrangement is that *z* represents depth; it is easy to imagine that one is looking into space behind the screen. The numbers produced by the cluster-simulation program are sometimes very large and sometimes very small. For this reason it is advisable to utilize double-precision arithmetic so that all relevant numbers are not inadvertently rounded.

The time taken by CLUSTER to finish one cycle of computation depends on the number of stars in one's system. As few as 10 stars will produce aesthetic intricacy; 100 or even 1,000 stars are needed to produce realistic complexity. Unfortunately the number of steps in the basic computational cycle increases as the square of the number of stars in the cluster. Although stellar simulators have found a neat method that dodges this particular limitation, other problems still arise.

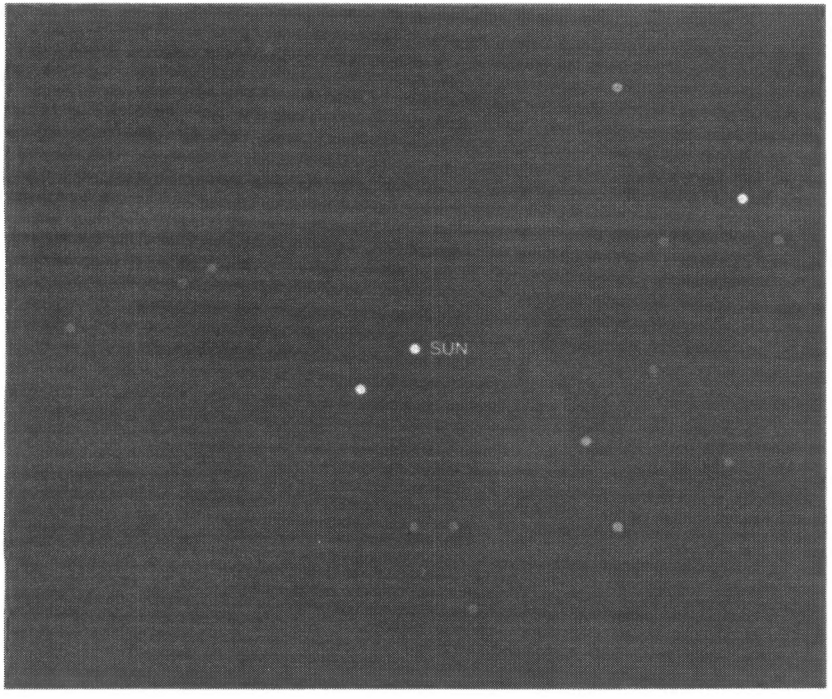
The worst problem emerges from the fact that the program is a discrete system attempting to mimic a continuous one. Continuous orbits are approximated by a sequence of jumps that depart increasingly from a star's true path through a cluster. The inaccuracy might be corrected to some extent by the presence of statistical regularities, but in close encounters between stars the system unnaturally and disastrously magnifies the slingshot effect.

For example, if the computational cycle puts one star (Stella) close to another star (Aster), a powerful gravitational pull magnifies the acceleration components of both stars. The magnification percolates through the computation to the velocity components and thence to the position coordinates. The next iteration finds Stella already widely separated from Aster and unable to repay the gravitational loan. A fiction of excess kinetic energy has been created. Artificial clusters afflicted by this problem evaporate even faster than real ones. There are two ways around the difficulty; one is hard, the other is easy. The hard alternative requires computation of a Keplerian orbit for the pair. The orbit is maintained as long as the two stars are in proximity. Theorists regard this as the method of choice because the orbital formula is perfectly accurate. An easy but occasionally inaccurate way to handle close encounters is to subdivide the time steps in the basic computational cycle. Readers may want to add this particular maneuver to the advanced version of CLUSTER that I shall now describe.

A program called SUPERCLUSTER can be derived from CLUSTER by a series of simple modifications. First, SUPERCLUSTER incorporates stars of different masses in its ballet. This is easily done at the start by entering the masses in an array called *m*. The force computation becomes somewhat more complicated: force is no longer proportional to $1/d^2$ but to the product of the masses divided by d^2. Next, SUPERCLUSTER incorporates spectral types. As in the case of mass, an array (called *spec*) must be filled in before the run. It is used, however, only during the display phase of the basic cycle. The colors range from blue for O-type stars to red for M types. Green is omitted. The third enhancement of CLUSTER makes arbitrary time steps possible in either version of the program.

SUPERCLUSTER uses a time-step variable called *delta*. Specified at the beginning of a run, *delta* determines the amount of simulated time between successive cycles. Naturally this time element must affect the updating formulas for both velocity and position: in the velocity formulas it multiplies acceleration and in the position formulas it multiplies velocity.

The easy way to handle close encounters can now be described. First a definition of "close" must be established. Then a test for such closeness can be inserted into the program just after the point at which the distance between two stars is calculated. If a close encounter is taking place, SUPERCLUSTER replaces *delta* by one-tenth of its value—at least until no pair of stars is that close again. This expedient certainly helps to cushion the sudden lurches of discrete gravity. It creates even worse problems when encounters are really close, however. An approach that is 10 times closer now results in a gravitational force that is 100



Would our galactic neighborhood form a cluster?

times greater! Fortunately close encounters of the worst kind are rather rare. The time-subdivision technique has been standard in cluster-simulation programs traditionally employed by professionals.

If SUPERCLUSTER is to be an astronomically meaningful program, units for distance, mass and other aspects of physical reality are needed. A convenient measure of distance is the astronomical unit (AU), which is equal to the earth's mean distance from the sun. Mass can be measured in solar masses and time is best measured in years. Under these conventions the universal constant of gravitation has the approximate value of 39. SUPERCLUSTER uses this constant instead of 1,000 in the force calculation.

All is now in readiness for putting either program to work. A preliminary exercise for CLUSTER involves four stars. Place them at the corners of a square that is an inch or two wide on the screen. It is only fair to give each star a nonzero z coordinate as well as the x and y coordinates that were mentioned above. If motion is confined to the plane of the screen, close encounters are that much commoner. Velocity components should be small (on the order of -5 to $+5$) and should specify a clockwise direction, as though the four stars were on a wheel.

SUPERCLUSTER can be tried on the system of stars shown on page 22. This is the earth's galactic neighborhood. What would happen if the sun and its neighboring stars were cut loose from our galaxy and allowed to dance endlessly in space? Would a cluster form? The question may or may not have scientific relevance, but it is fun to answer. Besides, these are the only stars for which positions and velocities are known accurately [*see illustration below*].

Clusters of stars are either open or globular. Open clusters consist of 1,000 or so stars, whereas globular clusters may consist of millions. So far investigators such as J. Garrett Jernigan at the University of California at Berkeley Space Sciences Laboratory have been able to handle only small clusters. Globular clusters are currently intractable. Even so, Jernigan and pioneering colleagues such as Sverre J. Aarseth of the University of California at Berkeley have been observing collapses of computer clusters for decades. The extent of collapse is measured by considering a spherical volume that is centered within a cluster and contains 10 percent of its mass. The radius of this volume is known as the 10 percent radius. Collapse is under way when the 10 percent radius decreases as time passes. Inexorably the core of a simulated cluster becomes ever denser. Since the simulated stars are mathematical points, nothing terrible ever happens to such clusters. No black hole comes into being at the center. This at least has been the experience of cluster theorists. But we seem able to find little evidence of extreme collapse in the clusters overhead. Something is preventing collapse out there.

Both traditional and modern simulation experiments may provide a key. On various occasions a small number of binary star systems at the center of a simulated cluster have brought the collapse of core regions virtually to a halt. In one of Jernigan's experiments a single binary seemed to be responsible. How is it possible? According to Jernigan's graduate student David Porter, it may be that "very tight binaries whizz around each other very quickly and kick wandering stars energetically around the core or even back out to a looser collection of stars around the core called the halo. This could be a mechanism for preventing the core from getting too crowded."

Jernigan used to be an observer of X-ray stars. As research focused on the search for X-ray sources in clusters, he grew increasingly interested in clusters as astronomical objects in their own right. Simulation seemed an effective way to investigate them.

Self-described as "the new kid on the block," Jernigan has discovered an important new efficiency in simulation efforts. In CLUSTER and similar programs a single computational cycle for n stars requires roughly n^2 steps. Jernigan's cycle needs only $n \times \log(n)$ steps. He organizes his cluster by grouping the stars into neighboring pairs. Each pair is then replaced by a fictitious mass and velocity that summarizes the behavior of the pair. The same process is now applied to the pairs as if they were the original stars. Continuing in this manner, a collection of grouped and regrouped mass nodes is built up in a data structure called a tree. The single node at its root simultaneously represents all the stars. Motions can then be calculated for the central node and

NAME OF STAR	POSITION COORDINATES			VELOCITY COORDINATES			COLOR	MASS
	X	Y	Z	VX	VY	VZ		
STRUVE 2398	68	−365	631	−5.69	4.76	3.35	RED	0.26
ROSS 248	464	−42	450	−8.75	1.13	−15.45	RED	0.17
61 CYGNI	394	−377	433	−2.78	22.03	0.02	ORANGE	0.69
LALANDE 21185	−404	107	307	7.32	−0.47	−20.11	RED	0.39
PROCYON 5	−295	658	68	2.38	0.75	−3.65	BLUE	1.29
BARNARD'S STAR	−7	−371	30	−0.87	24.20	16.78	RED	0.21
EPSILON ERIDANI	408	534	−114	4.60	0.69	−0.50	ORANGE	0.74
WOLF 359	−462	136	62	−0.82	9.86	−5.94	RED	0.10
SIRIUS	−98	514	−157	1.89	−2.21	−2.59	BLUE	2.96
LUYTEN 726–8	487	219	−175	2.08	10.80	−0.41	RED	0.19
ROSS 128	−683	44	13	2.51	−2.32	−4.09	RED	0.21
SUN	0	0	0	0.00	0.00	0.00	YELLOW	1.00
TAU CETI	646	307	−208	0.52	−6.62	3.92	YELLOW	0.85
ALPHA CENTAURI	−106	−86	−243	−1.95	4.68	4.51	YELLOW	1.03
LUYTEN 789–6	608	−235	−182	−6.75	10.81	10.56	RED	0.13
LUYTEN 725–32	718	227	−233	4.70	6.16	0.51	RED	0.21
ROSS 154	111	−536	−241	1.79	1.36	−0.11	RED	0.24
EPSILON INDI	334	−194	−594	−3.54	17.71	2.28	ORANGE	0.69

A table listing all but three stars in the neighborhood of our solar system

for all its branches out to the individual stars.

Is this the technique of the future? It certainly helps to speed things up, according to Jernigan. Yet subsequent generations of cluster programs are more likely to resemble the hybrid variety used by Alan P. Lightman of the Center for Astrophysics of the Harvard College Observatory and the Smithsonian Astrophysical Observatory and Stephen L. W. McMillan of the University of Illinois at Urbana-Champaign: stars in the core are handled by the direct simulation methods described above; stars outside the core are modeled statistically as if they form a gas.

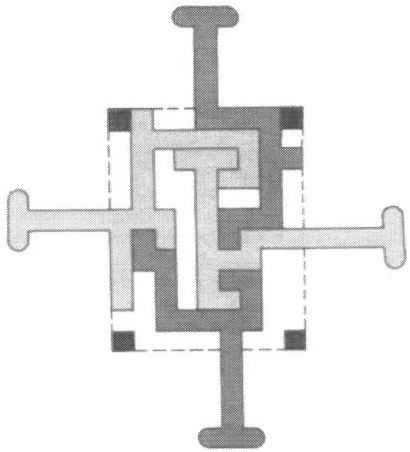
For readers proficient in the language called APL there is an interesting new publication by Gregory J. Chaitin of the IBM Thomas J. Watson Research Center in Yorktown Heights, N.Y. It is called *An APL2 Gallery of Mathematical Physics* and is a 56-page booklet containing explanations of five major physical theories, including those that describe both the Newtonian and the relativistic motion of satellites in space. APL listings are given for computer programs that illustrate each theory. Chaitin will be happy to send a copy to any reader who writes to him at the Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y. 10598.

In last October's column I described three puzzles: Bill's baffling burr, Coffin's cornucopia and Engel's enigma. Hundreds of readers have tackled the puzzles. While some seek the magic combination of moves that disassemble the burr, others scratch their head over the placement of polyominoes in a tray. Members of this group will have to get by without help from their friends: each puzzle is unique. Still other readers keep rotating the wheels of Engel's enigma in a vain attempt to unscramble it. Some of the devotees are succeeding, at least on an abstract plane: claims of solutions to the enigma have started to come in.

A call for two-dimensional burrs brought in a number of designs. The most charming design received so far is shown at the top of this page. The problem is to remove the four pieces from the tray symbolized by the rectangular outline. The pieces can only be moved in four directions confined to the plane of the page: up, down, left and right. The four corner squares are regarded as immovable. Which piece must be moved first? Jeffrey R. Carter of Littleton, Colo., designed this two-dimensional tour de force. Our three-dimensionality confers the advantage of visualizing the whole; a two-dimensional solver would push and pull at the sides of a mysterious box.

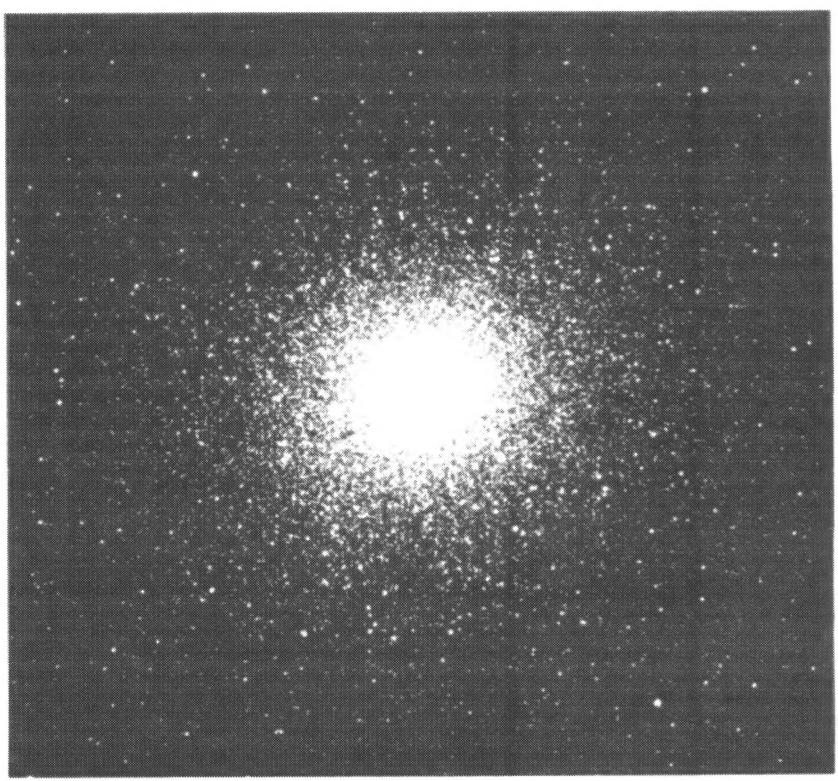
An algorithmic solution to Engel's enigma is claimed by P. Clavier of Dallas, Tex. Clavier says that his program, written in BASIC and running on a Texas Instruments CC-40 portable computer, solves typical scrambles in from 300 to 700 moves. The solution implements six fundamental exchange operations on the stones and bones. Readers who used the sequence representation I suggested may have cast their net too widely; solutions of the numerical sequence are not always solutions of the enigma. In framing the suggestion I was aware that bones were excluded from the representation. "Well," said I at the time, "take care of the stones and the bones will take care of themselves." Not so. The stones should be interleaved with the symbols that represent the bones.

The ultimate scramble-unscramble puzzle appears to have been invented by Robert Carlson of Los Altos, Calif. It is so complicated to make that he must be content with the view of it on his monitor. The puzzle is an icosahedron, the Platonic solid that consists of 20 triangular faces. Each vertex is the site of a possible scrambling operation.



Jeffrey R. Carter's two-dimensional burr

When a vertex is rotated, the five incident triangles are rotated as well. Each triangle has three colors. In unscrambled form the colors adjacent to each vertex are the same. Carlson has prepared a version of his computer puzzle for the IBM PC. In addition to colors it features a musical note for each move. Interested readers can obtain a disk by writing to Carlson at 319 Lunada Court, Los Altos, Calif. 95030.



Globular cluster Messier 13 in the constellation Hercules

In my January column I described two programs, CLUSTER and SUPERCLUSTER, that simulate the evolution of a star cluster. It heartens me to think that in at least a few thousand homes they have led to a new form of entertainment, temporarily edging out television. No doubt some of these armchair universes are unfolding as they should, but others may be developing problems. The fault is not in our stars but in ourselves.

Brian Davis of Ann Arbor, Mich., and Peter Fortescue of La Jolla, Calif., had trouble with the acceleration equations in SUPERCLUSTER. The difficulties are fixed, I believe, by replacing the force f in the equation on page 13 with the acceleration a due to the force. To get a divide f by the mass of the attracted star. Andrew M. Odlyzko of AT&T Bell Laboratories pointed out that the position coordinates in the table on page 15 are in multiples of 1,000 astronomical units (A.U.), not single A.U.'s. Our own universe will now unfold correctly.

When one views the live action of CLUSTER or SUPERCLUSTER on a display monitor, it is sometimes hard to tell which stars are in the foreground and which are farther back. Albert C. English of Delray Beach, Fla., and Peter Stearns of Lodi, Calif., have written special display programs that generate two images of clusters side by side, one as seen by the right eye and one as seen by the left. Readers able to manage the tricks of stereoscopic display will be able to view the clusters as they view the hypercube: in breathtaking depth.

Several readers had already written programs similar to SUPERCLUSTER, but they had applied the programs to our own solar system. The same application would also be feasible with SUPERCLUSTER. Those with the gumption can look up the mass, position and velocity of the 10 major bodies in the solar system for some reference time. One can then arrange to view the evolution of the entire system from above: wait a few minutes for the year 2000. Geoffrey L. Phillips of St. Louis, Mo., wrote a simulation for the earth-moon system that includes a small, massless space vehicle. Launching it from the earth in such a way that it begins to orbit the moon is no easy feat. Advanced practitioners might try launching a Voyager spacecraft on a grand tour of the gas giants that ends as it leaves the solar system.

William A. Hoff of Champaign, Ill., computed the time increment for the simulation dynamically by setting a variable called *dvmax* at the beginning of the program. In the course of the calculations of stellar motion the program always finds the maximum acceleration *amax* of a star. The next time increment is *dvmax* divided by *amax*. The technique prevents any velocity from exceeding *dvmax*.

In last month's column I gave an I.Q. minitest and posed several questions about numerical sequences. The first problem on the minitest is a good example of the ambiguity typically found in such problems. The problem was to complete the sequence 3, 7, 16, 35, ... Each term minus twice the preceding term gives the sequence 1, 2, 3, the second row of a pyramid. By this reasoning the missing term must be twice 35 plus 4, or 74. On the other hand, if a simple difference pyramid is constructed with three rows, the third row gives the sequence 5, 10, and it seems reasonable to complete the sequence with 15. The missing term must then be 69, but the programs described in the column would have missed this answer. The other answers to the test: H is the missing letter; the missing word is "up"; the odd man out is "identity"; the unscrambled name of the town not in Italy is Madrid, and the correct visual analogy is number 2.

The two numerical sequences on page 12 are completed by 350 and 22 respectively. The first sequence on page 10 can be solved by applying a generalized difference rule, with k equal to 3, and then a generalized quotient rule; the missing term is 324. The second sequence ought to defeat all but the most patient puzzle solvers who did not try to write SEQ. It can be solved by two quotient rules; the value of k in the first rule is 5. The missing term is $-65{,}551$.